OpenAI API Complete Study Guide

1. How OpenAI Models Work

Core Architecture

- **Transformer-based neural networks** trained on massive text datasets (trillions of tokens)

- **Autoregressive generation**: predicts the next token based on all previous tokens in sequence
- **Context window**: limited memory capacity (varies by model - GPT-3.5: ~4K tokens, GPT-4: up to 128K tokens)

Training Process

- **Pre-training**: learns patterns from internet text, books, articles
- **Fine-tuning**: specialized training on curated datasets
- **RLHF (Reinforcement Learning from Human Feedback)**: aligned with human preferences

How Inference Works

- Input text is **tokenized** into numerical representations
- Model calculates **probability distributions** for next possible tokens
- **Sampling strategies** (temperature, top-p) determine which token is selected
- Process repeats until stopping condition is met

Model Capabilities

- **Text generation**: creative writing, summaries, explanations
- **Code generation**: programming in multiple languages
- **Analysis**: sentiment analysis, data interpretation
- **Conversation**: maintaining context across multi-turn dialogues

Limitations

- **Knowledge cutoff**: training data has a specific end date
- **Hallucinations**: can generate plausible but incorrect information
- **Context limits**: cannot remember beyond token window
- **Stateless**: each API call is independent

2. Setting Up a Request

Basic Setup

```
import OpenAI from "openai";
```

```
const openai = new OpenAI({
    apiKey: "your-api-key",
    dangerouslyAllowBrowser: true  // Only for client-side
development
});
```

Essential Request Structure

```
const response = await openai.chat.completions.create({
    model: "gpt-4",
    messages: [
        {
            role: "system",
            content: "You are a helpful assistant."
        },
        {
            role: "user",
            content: "Hello, how are you?"
        }
    ]
});
```

Message Roles Explained

- **system**: Sets behavior and context (like giving instructions to the AI)
- **user**: Represents the human user's input
- **assistant**: The AI's responses (used for conversation history)
- **tool**: Used for function calling responses

Common Configuration Options

```
{
    model: "gpt-4",
    messages: messages,
    temperature: 1.0,         // Creativity level
    max_tokens: 150,          // Response length limit
    top_p: 1.0,              // Nucleus sampling
    frequency_penalty: 0,     // Reduce repetition
    presence_penalty: 0,      // Encourage topic diversity
    stop: ["\n"],            // Stop sequences
    stream: false           // Real-time streaming
}
```

Error Handling

```
try {
    const response = await openai.chat.completions.create({...});
    console.log(response.choices[0].message.content);
} catch (error) {
    console.error("API Error:", error.message);
    // Handle rate limits, invalid requests, etc.
}
```

3. Tokens

What Are Tokens?

- **Not words or characters** - chunks of text with varying lengths
- **Average**: approximately 4 characters per token in English
- **Examples**:
  - "hello" = 1 token
  - "ChatGPT" = 2 tokens
  - "artificial intelligence" = 2 tokens

Token Calculation

- Use OpenAI's tokenizer: https://platform.openai.com/tokenizer
- **Prompt tokens**: input message tokens
- **Completion tokens**: generated response tokens
- **Total tokens**: prompt + completion

Token Economics

- **Cost**: charged per token (input + output)
- **Context limits**: total conversation must fit within model's token limit
- **Performance**: fewer tokens = faster responses

Managing Tokens

```
// Limiting output length
max_tokens: 100  // Maximum tokens in response

// Checking token usage
console.log(response.usage);
// Output: {prompt_tokens: 44, completion_tokens: 56, total_tokens:
100}
```

Token Strategies

- **Summarization**: compress long conversations to stay within limits
- **Chunking**: break large documents into token-sized pieces
- **Optimization**: remove unnecessary words and formatting

4. Tools (Function Calling)

What Are Tools?

- **Function calling**: allows models to call external functions
- **Structured outputs**: get JSON responses for specific tasks
- **Chain operations**: combine multiple API calls intelligently

Setting Up Tools

```
const tools = [
    {
        type: "function",
        function: {
            name: "get_stock_price",
            description: "Get current stock price for a symbol",
            parameters: {
                type: "object",
                properties: {
                    symbol: {
                        type: "string",
                        description: "Stock symbol (e.g., AAPL,
TSLA)"
```

```
                    }
                },
                required: ["symbol"]
            }
        }
    }
];
```

Using Tools in Requests

```
const response = await openai.chat.completions.create({
    model: "gpt-4",
    messages: messages,
    tools: tools,
    tool_choice: "auto"  // or "none", or specific function
});

// Handle function calls
if (response.choices[0].message.tool_calls) {
    const toolCall = response.choices[0].message.tool_calls[0];
    const functionName = toolCall.function.name;
    const functionArgs = JSON.parse(toolCall.function.arguments);

    // Execute your function
    const result = await executeFunction(functionName,
functionArgs);

    // Send result back to model
    messages.push({
        role: "tool",
        content: JSON.stringify(result),
        tool_call_id: toolCall.id
    });
}
```

Tool Use Cases

- **API integrations**: weather, stocks, databases
- **Calculations**: mathematical operations
- **Data retrieval**: file systems, web scraping

- **External services**: email, SMS, payments

5. The "Few Shot" Approach

What Is Few-Shot Learning?

- **Learning from examples**: providing input-output pairs as guidance
- **Pattern recognition**: model learns desired format and style
- **No training required**: works immediately with examples

Basic Structure

```
const messages = [
    {
        role: "system",
        content: `You are a stock analyst. Format responses like the
examples below:

        ###
        Stock: AAPL
        Price: $150.25 (+2.3%)
        Recommendation: BUY
        Reason: Strong quarterly earnings
        ###

        Stock: MSFT
        Price: $280.50 (-1.2%)
        Recommendation: HOLD
        Reason: Market volatility concerns
        ###`
    },
    {
        role: "user",
        content: "Analyze TSLA stock"
    }
];
```

Few-Shot Strategies

- **Zero-shot**: no examples, just instructions
- **One-shot**: single example provided
- **Few-shot**: multiple examples (2-5 typically optimal)
- **Many-shot**: extensive examples (for complex tasks)

Best Practices

- **Clear delimiters**: use ### or --- to separate examples
- **Consistent format**: maintain same structure across examples
- **Diverse examples**: show different scenarios and edge cases
- **Quality over quantity**: better examples > more examples

Pros and Cons

Pros:

- More control over output format
- Consistent styling
- Works without fine-tuning
- Quick implementation

Cons:

- Uses more tokens (higher cost)
- Reduces available context space
- Can overfit to examples
- Less flexible than fine-tuning

6. Temperature

What Is Temperature?

- **Randomness control**: determines how "creative" or "conservative" responses are
- **Range**: 0 to 2 (default is 1)
- **Sampling parameter**: affects probability distribution of next token selection

Temperature Values

```
// Conservative (factual, predictable)
temperature: 0.2

// Balanced (default behavior)
temperature: 1.0

// Creative (diverse, unpredictable)
temperature: 1.8
```

Low Temperature (0 - 0.3)

- **Characteristics**: deterministic, factual, consistent
- **Use cases**:
    - Code generation
    - Mathematical calculations
    - Factual Q&A
    - Data analysis
    - Legal/medical content

Medium Temperature (0.4 - 1.2)

- **Characteristics**: balanced creativity and consistency
- **Use cases**:
    - General conversation
    - Content writing
    - Summaries
    - Explanations

High Temperature (1.3 - 2.0)

- **Characteristics**: highly creative, diverse, unpredictable
- **Use cases**:
    - Creative writing
    - Brainstorming
    - Poetry generation
    - Experimental content

Temperature vs Top-p

- **Alternative sampling methods**: use one or the other, not both at extremes
- **Temperature**: affects entire probability distribution
- **Top-p**: considers only top percentage of probable tokens

7. Frequency and Presence Penalties

Presence Penalty

What it does: Encourages discussing new topics **Range**: -2.0 to 2.0 (default: 0) **How it works**: Penalizes tokens that have already appeared

```
presence_penalty: 0.6  // Encourages topic diversity
```

Effects:

- **Low (0 to 0.5)**: Focuses on current topics
- **Medium (0.5 to 1.0)**: Balanced topic exploration
- **High (1.0 to 2.0)**: Jumps between many topics

Use cases:

- **Low**: Detailed analysis of single topic
- **High**: Brainstorming multiple ideas

Frequency Penalty

What it does: Reduces repetition of the same phrases **Range**: -2.0 to 2.0 (default: 0) **How it works**: Penalizes tokens based on frequency of use

```
frequency_penalty: 0.8  // Reduces repetitive language
```

Effects:

- **Low (0 to 0.5)**: May repeat key phrases
- **Medium (0.5 to 1.0)**: Varied language
- **High (1.0 to 2.0)**: Avoids repetition strongly

Use cases:

- **Low**: Technical documentation (repetition needed)
- **High**: Creative writing (variety preferred)

Combined Usage

```
{
    temperature: 0.8,
    presence_penalty: 0.4,   // Slight topic diversity
    frequency_penalty: 0.6   // Reduce repetition
}
```

Practical Examples

For stock analysis:

```
// Focused, consistent analysis
{
    temperature: 0.3,
    presence_penalty: 0.1,
    frequency_penalty: 0.2
}

// Creative, varied reporting
{
    temperature: 1.1,
    presence_penalty: 0.6,
    frequency_penalty: 0.8
}
```

Best Practices

- **Start with defaults** (all at 0) and adjust incrementally
- **Test combinations** - penalties interact with temperature
- **Monitor output quality** - high penalties can hurt coherence
- **Use moderately** - extreme values (>1.5) often degrade performance
- **Consider your use case** - factual content needs different settings than creative content